Paralelización del código Stampack v7.10

W.Castelló
F. Flores

Publicación CIMNE Nº-362, Julio 2011

Paralelización del código Stampack v7.10

W. Castelló
F. Flores

Publicación CIMNE Nº-362, Julio 2011

Centro Internacional de Métodos Numéricos en Ingeniería
Gran Capitán s/n, 08034 Barcelona, España

Índice

1. Introducción

Hoy en día las computadoras paralelas o *clusters* de ordenadores son muy comunes en los centros de investigación, algunas empresas grandes cuentan también con ellas, y son ampliamente utilizados para realizar cálculos complejos o extensos. Un desafío en el cálculo en paralelo es el desarrollo de códigos capaces de utilizar eficientemente las capacidades del hardware disponible para resolver problemas más grandes en menos tiempo. Pero la programación en paralelo no es una tarea fácil, y además existen diversas arquitecturas que pueden elegirse. Se puede agrupar las arquitecturas de datos en paralelo en dos familias principalmente:

Arquitectura de memoria compartida: en este tipo de máquinas en paralelo se puede observar un conjunto de procesadores que tienen acceso a una memoria en común. Por lo general a los equipos basados en esta arquitectura se los denomina ordenadores SMP, donde SMP son las siglas en ingles de multiprocesamiento simétrico.

Arquitectura de memoria distribuida: en estas máquinas por otra parte cada procesador tiene su propia memoria privada y la información se intercambia entre los procesadores a través de mensajes. El nombre de *clusters* esta asociado comúnmente a este tipo de dispositivos informáticos y el tipo de lenguaje de intercambio de datos mas común es denominado MPI (message passing interface).

Cada una de estas dos familias tiene sus ventajas y desventajas; y las normas actuales de programación paralela intentar aprovechar al máximo estas ventajas centrándose en alguna de estas dos arquitecturas. En los últimos años se creó y desarrolló un nuevo estándar, cuyo objetivo es servir como una buena base para el desarrollo de programas paralelos en las máquinas de memoria compartida: OpenMP.

OpenMP (Open Multi Processing) es el resultado de un acuerdo entre los proveedores de hardware y desarrolladores de compiladores; es considerado como un "estándar de la industria" y contiene: un conjunto de directivas para el compilador, una biblioteca de rutinas, y las variables de entorno necesarias para especificar la paralelización empleando memoria compartida en Fortran y C / C++. OpenMP consolida todo esto en una sola sintaxis empleando una semántica muy simple.

Las ventajas que supone el uso de OpenMP en el código Stampack, frente a otros tipos de arquitecturas de paralelización, son básicamente dos:

1. La sintaxis y la semántica de las directivas y las subrutinas de paralelización definidas en OpenMP implican una mínima intervención sobre el código fuente actual, es decir que las modificaciones al código de Stampack no suponen cambios importantes y además permite mantener la opción original de cálculo serial.

2. Las empresas de hardware han volcado su desarrollo a la explotación de los ordenadores multi-núcleo o multi-trheading, los cuales funcionan sobre una base de memoria compartida (por ej.: los procesadores de la familia Intel Core i3, i5 e i7). Es decir resulta natural el uso de OpenMP en este tipo de ordenadores personales, y se justifica en el hecho de usar eficientemente todos los procesadores del ordenador para reducir el tiempo de cálculo.

El contenido de este informe se resume a continuación. En la Sección 2 se presenta el tipo de ordenador objetivo en el cual se ha maximizado su potencia de cálculo empleando OpenMP. En la Sección 3 se discuten los aspectos generales de la implementación de

OpenMP en el código Stampack, además se presentan algunos inconvenientes encontrados en la paralelizacion y las soluciones a los mismos, por último en esta sección se enumeran las rutinas que contienen las modificaciones empleando la sintaxis de OpenMP. La Sección 4 muestra algunos resultados comparativos en problemas significativos, los resultados muestran la importante disminución en el tiempo de cálculo cuando se compara un código secuencial con uno en paralelo. Para finalizar las Secciones 5 y 6 muestran las tareas por hacer y un resumen de las conclusiones obtenidas de este informe, respectivamente.

2. Objetivos de la paralelización

El objetivo principal del presente informe es mostrar las mejoras en la *performance* del código Stampack, cuando se lo emplea en conjunto con OpenMP en máquinas del tipo multi-núcleo (ej.: Intel Core2 Duo, Intel Core2 Quad, etc) o *multi-thread* (ej.: Intel Core i3, i5, i7, etc). Esta última tecnología en los procesadores (*multi-thread*) permite aumentar de manera importante la potencia de cálculo, dado que además de contar con más de un procesador, permite aumentar los hilos (*threads*) de tareas simultaneas de cada núcleo. Por ejemplo un procesador Intel Core i3, puede tener dos procesadores y realizar dos tareas simultaneas por cada procesador, lo cual es equivalente a tener una máquina con cuatro procesadores. En el caso del procesador Intel Core i7 empleado en este trabajo cuenta con cuatro núcleos y la posibilidad de realizar dos tareas simultaneas por núcleo, lo cual equivale a tener ocho procesadores.

La paralelización del software Stampack se ha realizado siguiendo la idea básica de modificar o intervenir mínimamente los archivos fuente originales de la versión serial. Bajo estas condiciones, se ha realizado un análisis de tiempos para detectar zonas del código que representen un importante porcentaje del tiempo total de cálculo.

Para finalizar se debe destacar que el software paralelizado debe funcionar también de manera correcta cuando es ejecutado en ordenadores con un solo procesador, de manera que la versión paralela debe ser compatible en su totalidad con la versión serial original del código Stampack.

3. Implementación del esquema en paralelo

El análisis de tiempos parciales durante el proceso total de cálculo en una simulacion de problemas de conformado, muestra que la evaluación de la contribución elemental sobre el vector residuo (bucle sobre los elementos) demanda en promedio un 85 % del tiempo total de cálculo. Por este motivo la primer zona a modificar es bucle sobre el conjunto de elementos, donde se evalúa el aporte de cada uno de ellos al residuo del problema, y en particular ello representa la intervención de los archivos fuentes `resvXX.fi` (donde XX hace referencia al número que identifica a cada elemento disponible en Stampack).

Es importante decir también que la paralelizacion alcanza actualmente a los elementos de Stampack que son utilizados con mayor frecuencia en la simulacion de procesos de conformado de solidos (bi y tridimensionales) y en estampado de laminas.

A continuación se muestra el esquema general de paralelización que se repite en los archivos fuente mencionados previamente, es conveniente destacar que el esquema general no cambia de un elemento a otro, aunque se debe prestar especial atención al hecho de que cada elemento puede tener algún tratamiento especial cuando se tienen bucles anidados (por ejemplo, bucles de evaluación del residuo a nivel puntos de Gauss dentro del bucle a nivel elemental).

En este apartado también se describen los inconvenientes encontrados en la paralelización, y además cuales son las rutinas del código Stampack que han sido modificadas hasta este momento.

3.1. Esquema general de paralelización

El esquema general de paralelización (similar al utilizado en versiones anteriores) se corresponde con el cuadro mostrado en la Figura 1, básicamente se define una zona donde el código se ejecuta en paralelo (PARALLEL-END PARALLEL). En esta zona el total de los elementos se dividen en `N` cantidad de bloques o conjuntos de elementos que serán ejecutados por `N` *threads* (el tamaño de estos bloques se hace comúnmente de manera que todos los hilos tengan un trabajo similar).

```
!$OMP PARALLEL                        &
!$OMP SHARED (...,variables,...)      &
!$OMP PRIVATE(...,variables,...)

  ... división del total de elementos en N bloques
      para que sean ejecutados por N threads ...

  DO elem = ielem,felem

    ... cálculos a nivel elemental ...

    DO gp = 1,ngaus

      ... cálculos a nivel puntos de gauss ...

    END DO

    !$ CALL omp_set_lock(lockvar)

     ... volcado de la contribución al vector residuo ...

    !$ CALL omp_set_lock(lockvar)

  END DO

!OMP END PARALLEL
```

Figura 1: Esquema básico de paralelización las rutinas de residuo a nivel elemental.

La zona que se ha paralelizado contiene variables declaradas como privadas (PRIVATE) o compartidas (SHARED), en el caso de las variables compartidas las mismas están asociadas a aquellas variables que no deben ser tratadas en paralelo y responden principalmente a variables comunes de donde se extraen datos generales (ej.: coordenadas, punteros a bases de datos, etc.) o bien a aquellas variables que deben ser actualizadas una vez calculada la contribución a nivel elemental (ej.: vector residuo). Por otra parte las variables privadas son aquellas no incluidas en el grupo anterior y en este caso cada *thread* puede tener asociado para cada la variable un valor diferente (ej: índices de bucles,

variables a nivel elemental, etc.). Puede decirse que toda variable global (ya sea un argumento DUMMY o perteneciente a un módulo) es compartida y prácticamente el total de las variables locales son privadas. De esta forma el espacio de memoria privada de cada hilo se restringe a variables locales que dependen de las características del elemento pero nunca depende del tamaño de la discretización.

El código Stampack emplea listas unidas (punteros) para definir las variables asociadas a cada elemento, debido a esto es necesario encontrar el puntero que asigna a cada thread la dirección del primer elemento de cada conjunto. Esta tarea se realiza previo al bucle en paralelo sobre los elementos.

Una tarea crítica en la paralelización de códigos computacionales es la actualización de variables compartidas (principalmente el vector de residuo en el caso de este informe). Existen diferentes problemas que pueden aparecer en códigos paralelos que no se presentan en la versión secuencial o en serie. En particular un inconveniente serio es la condición de carrera o *race condition*, esto ocurre debido a que cada *thread* puede hacer modificaciones sobre una variable especifica que no ha sido actualizada correctamente, si la ejecución en paralelo no se hace de manera ordenada. La solución a este problema es utilizar alguna sentencia de control (ej.: CRITICAL, ATOMIC, etc.) o bien alguna subrutina de OpenMP (ej.: *locks*). Desde un punto de vista de eficiencia las subrutinas de bloqueo o locks son mejores puesto que flexibilizan el trabajo en paralelo. La sentencia CRITICAL restringe el acceso a la zona definida como crítica a un solo *thread* por vez, mientras que en el caso de las subrutinas de bloqueo restringen el acceso a una parte de la memoria a un solo *thread* por vez. Para lograr este objetivo las subrutinas de bloqueo están asociadas a una variable que oficia de llave (`lockvar` en la Figura 1), en donde cada llave esta ligada a una posición (o varias posiciones) en la memoria y una vez que es tomada por un *thread* no puede ser empleada por otro *thread* hasta que el primero la libere.

3.2. Inconvenientes encontrados durante la paralelización

La programación en paralelo involucra distintos tipos de inconvenientes, en particular asociados a la forma en que se desarrollan las tareas en paralelo, la cual es muy diferente a la forma secuencial o serial. Las hilos o *threads* avanzan de manera independiente sobre el código paralelizado y esto puede acarrear inconvenientes: (a) la ya mencionada condición de carrera, (b) las subrutinas empleadas en el código deben ser del tipo tarea-segura o *thread-safe*, y (c) se debe tratar con cuidado la división del bloque de operaciones que debe ejecutar cada *thread*. A continuación se presentan los inconvenientes encontrados durante la paralelizacion del código Stampack y la propuesta de solución adoptada en cada caso.

3.2.1. Definición de subrutinas del tipo tarea-segura (thread-safe)

En los códigos secuenciales la llamada a subrutinas no evidencia inconvenientes dado que cada subrutina es ejecutada de manera ordenada por un solo hilo o *thread*. En el caso de los códigos con zonas paralelizadas lo expuesto no es necesariamente cierto, de hecho las subrutinas llamadas desde zonas del código con cálculo en paralelo pueden ser ejecutadas por mas de un hilo a la vez. Esto implica que se debe prestar especial atención a estas subrutinas, de manera que no exista pasaje o volcado de información entre los hilos o *threads* que están ejecutando la subrutina al mismo tiempo.

Para lograr que una subrutina sea del tipo tarea-segura existen como mínimo tres conceptos que deben respetarse: (1) las variables empleadas dentro de las subrutinas deben haber sido definidas correctamente como PRIVATE (privadas) o SHARED (compartidas)

-teniendo en cuenta la descripción previa de estas definiciones- en la zona paralela del código que llama a la subrutina en cuestión, (2) evitar en lo posible el uso de variables de definición o alcance GLOBAL dentro de la subrutina en cuestión, y (3) las subrutinas no deben incluir variables "estáticas", por lo cual no puede haber variables con atributo SAVE, sino que todas las variables han de ser "dinámicas" para que formen parte del STACK de cada hilo.

En particular el atributo SAVE tiene el inconveniente de que favorece el volcado de información entre los hilos. En el caso de códigos seriales, el uso de la sentencia SAVE permite guardar algunas variables dentro de la subrutina a los fines de no tener que evaluarlas continuamente si las mismas no cambian. Sin embargo este tratamiento de las variables trae aparejado que un hilo o *thread* tome el valor de una variable que ha sido guardada por otro hilo.

En el código Stampack hay muchas variables que tienen el atributo SAVE. Las que son de alcance global se mantienen con dicho atributo ya que permiten inicialización de variables y específicamente al ser usadas serán de tipo compartido. Por otro lado en aquellas variables locales en rutinas que han de llamarse desde una zona paralelizada, el atributo SAVE ha sido eliminado. A futuro podría pensarse en usar la sentencia de compilación opcional de OpenMP (!$) para sacar la ventaja del atributo SAVE cuando el código se ejecuta de manera secuencial, y evitar los problemas que acarrea cuando se lo ejecuta de manera paralela.

3.2.2. Incompatibilidad de la paralelización frente a problemas con grandes cantidades de elementos

El estándar de paralelizacion OpenMP posee diferentes métodos de división de conjuntos de tareas para los distintos hilos o threads, (a) estática: en donde se define el tamaño del bloque de tareas que toma cada hilo o *thread* y este no se modifica durante la ejecución, (b) dinámico: donde se establece de manera dinámica el bloque de tareas que toma cada hilo y este puede modificarse durante la ejecución, y (c) guiado: donde no solo se modifica el tamaño del bloque de tareas sino que además se modifica la cantidad de hilos que participan en la ejecución.

Actualmente en el código Stampack se emplea un esquema del tipo estático, en donde desde el inicio se divide en partes iguales el número de elementos que cada hilo procesa. Esto tiene un inconveniente y un problema. El inconveniente es que debido a los modelos constitutivos y al nivel de deformación no todos los elementos demandan la misma cantidad de operaciones, por lo cual algunos hilos pueden encontrarse sobrecargados en relación a los otros. A futuro se pretende usar un esquema del tipo dinámico, donde cada hilo tome parte del trabajo de manera ordenada y progresiva, para lo cual debe pensarse en la forma de trabajar en el arreglo de elementos donde actualmente se emplea una variable del tipo puntero. Por otro lado debido a la forma que habitualmente se ordenan los elementos en la generación automática de mallas esto normalmente implica mayores esperas al realizar el volcado sobre el vector de fuerzas residuales. El problema es que el tamaño del bloque de tareas a ser ejecutado por cada hilo puede exceder el tamaño de datos que puede manejar coherentemente el hilo o *thread*, generando un fallo del sistema (se manifiesta como un problema similar al volcado de pila) y la consecuente detención en la ejecución del código. Este problema (que se manifiesta cuando la cantidad de elementos es importante en malla finas) obliga a utilizar un bucle externo al de paralelización donde se limita la cantidad de elementos asignados a cada hilo, y el esquema general modificado resulta el que observa en la Figura 2.

```
   DO iblock = 1,nblock

  ... se divide el total de elementos de acuerdo a la
      cantidad que puede manejar coherentemente cada thread ...

!$OMP PARALLEL                      &
!$OMP SHARED (...,variables,...)    &
!$OMP PRIVATE(...,variables,...)

  ... división del total de elementos en N bloques
      para que sean ejecutados por N threads ...

  DO elem = ielem,felem

     ... cálculos a nivel elemental ...

    DO gp = 1,ngaus

       ... cálculos a nivel puntos de gauss ...

    END DO

    !$ CALL omp_set_lock(lockvar)

     ... volcado de la contribución al vector residuo ...

    !$ CALL omp_set_lock(lockvar)

  END DO
!OMP END PARALLEL

  END DO
```

Figura 2: Esquema modificado de paralelización las rutinas de residuo a nivel elemental.

3.3. Subrutinas con aplicación en paralelo

El esquema general mostrado en la Figura 2, se ha aplicado a distintos elementos dentro de Stampack. Estos elementos se corresponden con los problemas de conformado plástico mas comunes realizados con el código mencionado. Los elementos en cuestión corresponden a: (1) análisis de conformado de solidos bidimensionales (TR2D, elemento 20), (2) análisis de conformado de solidos tridimensionales o laminas considerando las mismas como solidos tridimensionales (SOLAG, elemento 18), y (3) análisis de conformado de laminas (NBST y LBST, elementos 13 y 14).

Las subrutinas que presentan actualmente la zona de cálculo en paralelo son:

- resv20.fi
- resv18.fi

- resv13.fi
- resv14.fi

Se puede observar que la paralelizacion solo involucra en este caso cuatro rutinas del código asociados a la evaluación de fuerzas nodales equivalente, esto demuestra que la intervención al código fuente es mínima. Además del control de las rutinas llamadas desde allí por supuesto. Por otro lado se han realizado algunas intervenciones en la la rutina de integración (explit.f90) y a futuro se podrían intervenir algunas otras áreas del código como la evaluación de las fuerzas de contacto, la actualización de las aceleraciones y velocidades, etc.

4. Resultados Numéricos

Se presentan a continuación los resultados obtenidos para algunos casos significativos desde el punto de vista de la paralelización. Los ejemplos presentados son relativamente pequeños comparados con aplicaciones industriales, la razón de ello es poder correr el mismo ejemplo en distintas arquitecturas y con distintas versiones del código en tiempos relativamente cortos. Por otro lado se han realizado corridas de modelos típicos industriales a los fines de verificar el correcto funcionamiento del código.

4.1. Embutición profunda de una lámina cuadrada (butter box)

Este problema involucra la embutición profunda de una lámina cuadrada para obtener un recipiente cuya geometría es similar a una caja de mantequilla (butter box). Las herramientas empleadas, el punzón, la matriz y el pisador, se consideran como solidos rígidos. La lamina puede modelarse a partir de elementos de lamina (shells) o bien con elementos solidos. Para el caso de elementos de lamina se han empleado elementos LBST (elemento de lamina delgada triangular para grandes deformaciones) y NBST (elemento de lamina delgada triangular en grandes deformaciones para geometrías no suaves o ramificadas). Por otra parte para el modelo de elementos solidos se ha empleado el elemento SOLAG (elemento hexaedro con deformaciones logarítmicas y una formulación lagrangiana), este elemento cuenta con la ventaja de poder usar *escalamiento selectivo de masas* lo cual permite aumentar el tiempo critico en la integración de las ecuaciones de movimiento y por ende disminuir los tiempos de análisis.

La geometría del problema es la que se define en la Figura 3, en esta figura pueden verse la posición inicial de la lamina y las herramientas. El material de la lamina es un acero cuyo módulo de Young es $E = 2{,}06 \times 10^{11}\text{Pa}$, un módulo de Poisson $\nu = 0{,}3$ y una densidad $\rho = 7{,}8 \times 10^{3}\text{kg/m}^3$. El comportamiento plástico del material está definido por una ley del tipo Ludwik-Nadai, donde la tensión de fluencia resulta y el módulo de endurecimiento varían según

$$\sigma_y = 567{,}29 \times 10^{6}(7{,}127 \times 10^{-3} + \varepsilon_p)^{0{,}2637}\text{Pa}$$
$$A' = 567{,}29 \times 10^{6} \cdot 0{,}2637(7{,}127 \times 10^{-3} + \varepsilon_p)^{0{,}2637-1}\text{Pa}$$

En el caso del análisis con elementos de lámina (LBST y NBST) se han empleado 1800 elementos triangulares de tres nodos, mientras que en el caso del análisis con elementos de sólido (SOLAG) se han empleado también 3600 elementos hexaedro definidos por ocho nodos (4 capas de elementos en el espesor de la lámina).

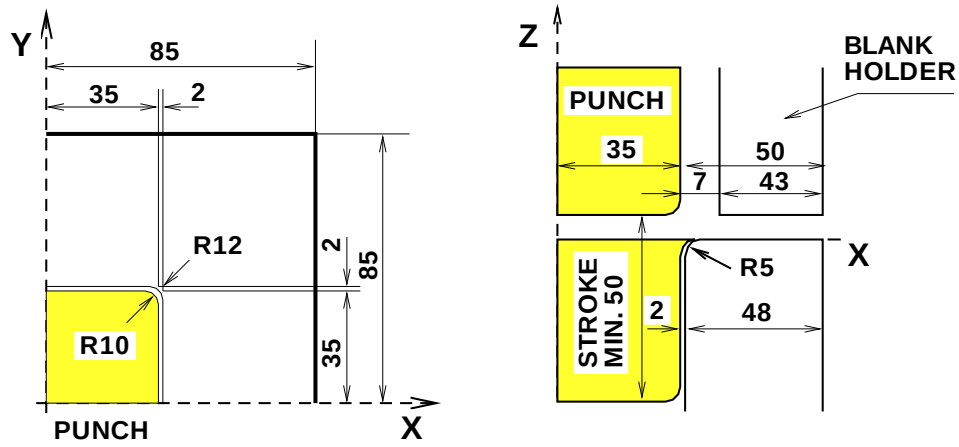
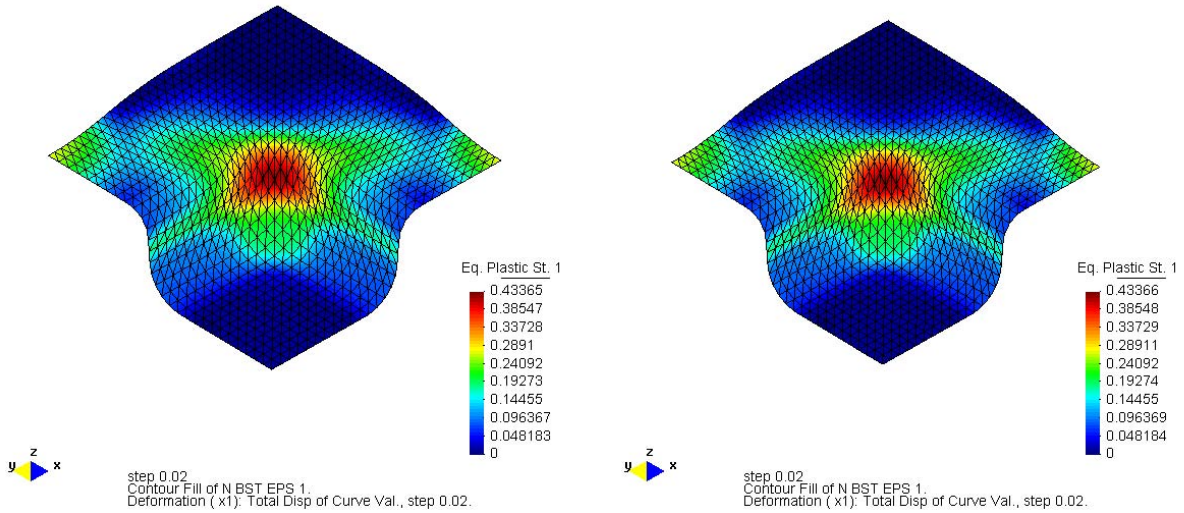


Figura 3: Geometría del problema de embutición de una lámina cuadrada.

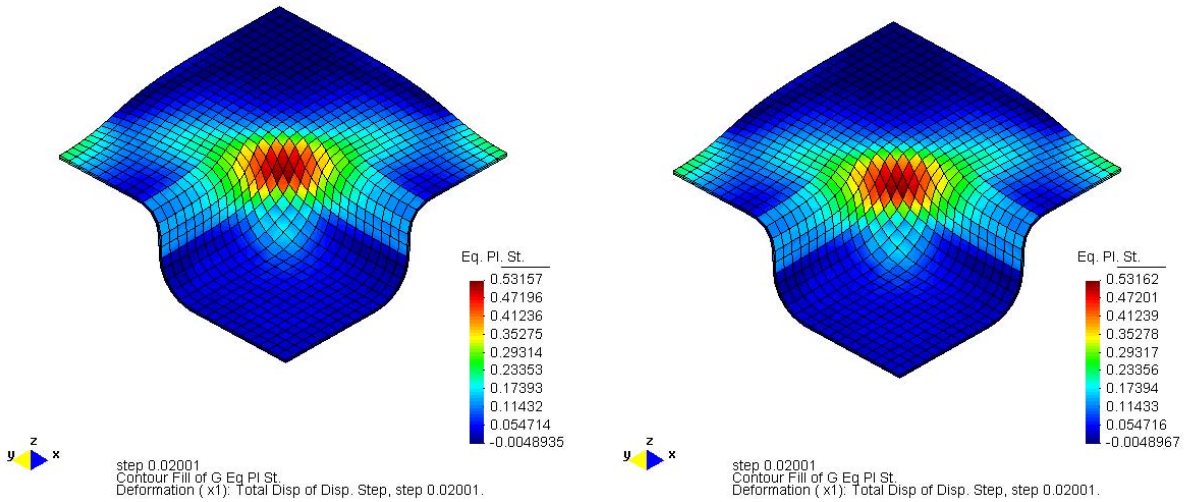
La comparación de resultados muestra concordancia, independientemente de: que tipo de elementos sean utilizados para el análisis del problema, y de que estrategia de ejecución se adopte (serie o paralelo). La Figura 4 muestra una comparativa de los resultados obtenidos para la deformación plástica efectiva en los puntos de gauss con elementos de lámina (NBST), mientras la Figura 5 muestra los mismos resultados en la embutición considerando elementos sólidos (SOLAG). Se debe destacar que se ha considerado para la comparación un posición de avance del punzón de 20 mm. Las diferencias entre los elementos de lámina y sólido están asociados a las diferencias propias de las formulaciones empleadas en cada elemento.



(a) **(b)**

Figura 4: Deformación plástica efectiva en los puntos de gauss empleando elementos de lámina NBST. (a) Serie. (b) Paralelo.

Los resultados que se presentan a continuación responden a corridas realizadas en tres distintos ordenadores: (a) un procesador Intel Core 2 Duo de dos núcleos y 2.66 GHz de velocidad en cada núcleo (2 threads), (b) un procesador Intel Core i5-750 de cuatro



(a) **(b)**

Figura 5: Deformación plástica efectiva en los puntos de gauss empleando elementos de sólido SOLAG. (a) Serie. (b) Paralelo.

núcleos y 2.8 GHz de velocidad en cada núcleo (4 threads), y (c) un procesador Intel Core i7-2600 de cuatro núcleos y 3.4 GHz de velocidad en cada núcleo (8 threads).

La Tabla 1 muestra los tiempos de cálculo asociados a cada ordenador ejecutando la versión en paralelo de Stampack, además se ha ejecutado cada problema con la versión serie a fin de poder visualizar la eficiencia del computo en paralelo.

Elemento	**LBST**		**NBST**		**SOLAG**	
Ejecución	serie	paralelo	serie	paralelo	serie	paralelo
Core 2 Duo	221.50 seg	155.80 seg	267.50 seg	180.60 seg	415.80 seg	289.00 seg
Core i5	154.00 seg	69.15 seg	182.50 seg	76.74 seg	290.40 seg	113.70 seg
Core i7	115.00 seg	49.05 seg	137.40 seg	50.92 seg	221.50 seg	73.68 seg

Tabla 1: Comparativa de tiempos de cálculo en ejecutables del tipo serie y paralelo en el problema del butter box.

Además la Tabla 2 muestra los porcentuales del tiempo utilizados en la versión paralela respecto a la serial. Los resultados muestran, una relación cercana a la lineal con el aumento de cantidad de núcleos en el caso del procesadores Core 2 Duo y Core i5. Sin embargo se debe tener en cuenta que en la disminución de tiempos no solo participa la cantidad de núcleos empleados, también juega un rol importante la velocidad de los procesadores.

Esta tabla sólo relaciona los tiempos totales y no muestra dos aspectos importantes

- que sólo se ha paralizado el bucle de fuerzas residuales
- que al compilar con OpenMP las rutinas no paralelizadas muestran un deterioro en su performance

Si sólo se comparan los tiempos dedicados a las fuerzas residuales los porcentuales mejoran mucho (la mejora es mayor con mayor número de threads). En la tabla 3 se muestran

Elemento	LBST	NBST	SOLAG
Core 2 Duo	70.33 %	67.51 %	69.05 %
Core i5	44.90 %	42.05 %	39.15 %
Core i7	42.65 %	37.06 %	33.26 %

Tabla 2: Relación de tiempos porcentuales asociados a la paralelizacion en el problema del butter box.

estos porcentajes. La Figura 6 muestra de forma grafica estos resultados, las barras corresponden a la relacion de tiempos porcentuales cuando se paraleliza el problema y se considera el calculo de las fuerzas residuales (el 100 % corresponde al tiempo serial).

Elemento	LBST	NBST	SOLAG
Core 2 Duo	66.87 %	64.62 %	66.79 %
Core i5	36.20 %	35.31 %	32.39 %
Core i7	30.09 %	26.17 %	24.01 %

Tabla 3: Relación de tiempos porcentuales asociados a la paralelización en el problema del butter box, considerando sólo cálculo de fuerzas residuales.

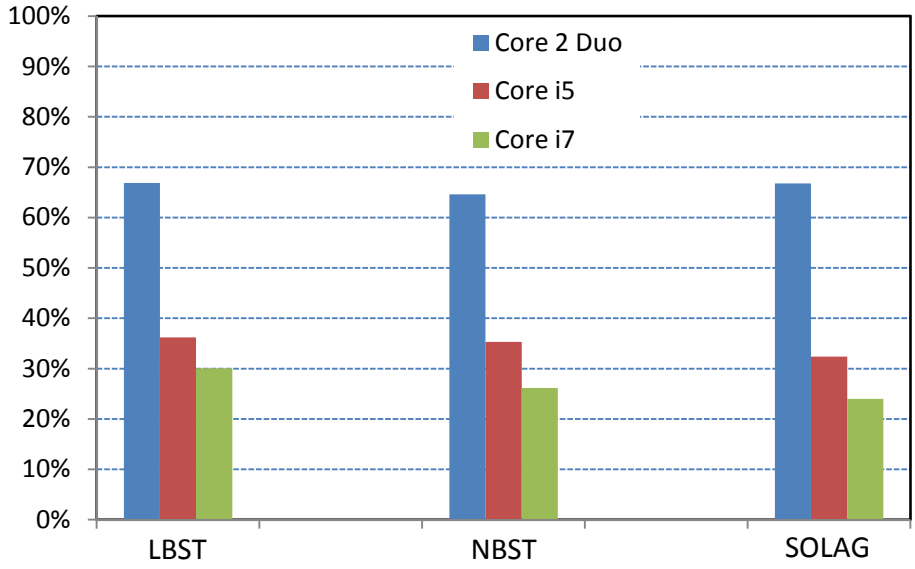


Figura 6: Relación de tiempos porcentuales asociados a la paralelización en el problema del butter box, considerando sólo cálculo de fuerzas residuales.

4.2. Acortamiento de un tocho cilíndrico

Este problema consiste en la compresión de un tocho cilíndrico para reducir su longitud hasta un 40 % de su longitud original. Este análisis involucra grandes deformaciones y además una importante distorsión de la malla. Se estudia un pequeño tocho cilíndrico de 30 mm de longitud, con un radio de 10 mm, ubicado entre dos placas rígidas perfectamente rugosas. El modelo del material está definido por un módulo de Young $E = 200$ GPa, una relación de Poisson $\nu = 0{,}3$ y una densidad de $\rho = 7833$ kg/m^3. Se asume un

endurecimiento isótropo, con una tensión de fluencia $\sigma_y = 700$ MPa y un módulo de endurecimiento $A' = 0{,}3$ GPa. La fricción entre la prensa y el tocho se modela con un coeficiente de fricción = 1.00, lo que implica que no hay deslizamiento entre el cuerpo y la herramienta.

La Figura 7 muestra la geometría del problema inicial del problema y la geometría deformada final. Se han empleado 14700 elementos triangulares. Este problema se caracteriza por presentar tiempos críticos muy pequeños, mas aun en la medida que avanza la deformación del tocho.

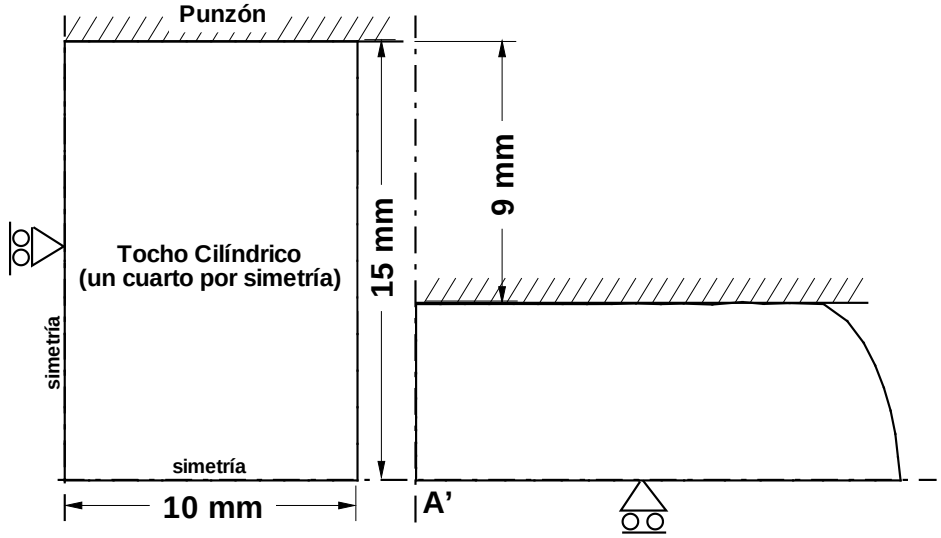


Figura 7: Geometría original y deformada del problema del acortamiento del tocho cilíndrico.

En este caso también la comparación de resultados muestra concordancia, independientemente de la estrategia de ejecución en el análisis del problema. Los resultados presentado, de manera similar al caso anterior, corresponden a corridas realizadas en tres distintos ordenadores: (a) un procesador Intel Core 2 Duo de dos núcleos y 2.66 GHz de velocidad en cada núcleo (2 threads), (b) un procesador Intel Core i5 de cuatro núcleos y 2.8 GHz de velocidad en cada núcleo (4 threads), y (c) un procesador Intel Core i7 de cuatro núcleos y 3.4 GHz de velocidad en cada núcleo (8 threads).

Ejecución	serie	paralelo	%
Core 2 Duo	7259 seg	5935 seg	81.76
Core i5	4788 seg	2333 seg	48.73
Core i7	3810 seg	1490 seg	39.11

Tabla 4: Tiempos de cálculo en el problema del tocho cilíndrico.

La Tabla 4 muestra los resultados de este caso, y se observa una disminución porcentual importante de los tiempos de cálculo. Por otro lado ne la Tabla 5 se muetran los tiempos correspondientes sólo a la evaluación de fuerzas residuales. Sin embargo, en este problema la incidencia de la paralelizacion no resulta de la misma magnitud que en el problema anterior. La Figura 8 muestra de forma grafica estos resultados, las barras corresponden a la relación de tiempos porcentuales cuando se paraleliza el problema y se considera el calculo de las fuerzas residuales (el 100 % corresponde al tiempo serial).

Ejecución	serie	paralelo	%
Core 2 Duo	6839 seg	5494 seg	80.33
Core i5	4480 seg	1975 seg	44.08
Core i7	3599 seg	1175 seg	32.68

Tabla 5: Tiempos de cálculo de fuerzas residuales en el problema del tocho cilindrico.

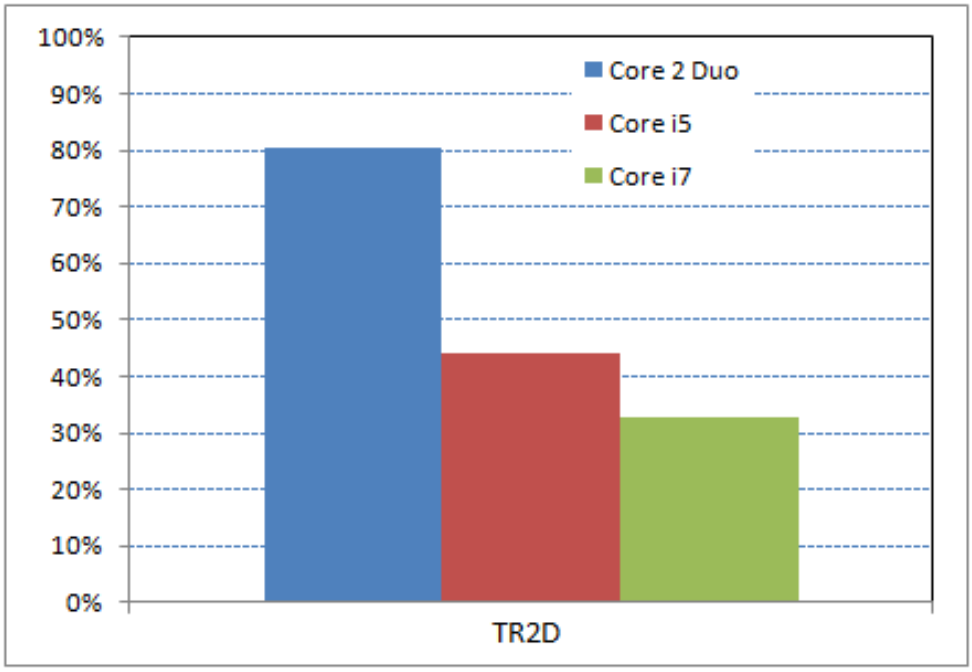


Figura 8: Tiempos de cálculo de fuerzas residuales en el problema del tocho cilíndrico.

5. Tareas por realizar

Un aspecto que debe solucionarse, a nivel de compilación y con las opciones adecuadas, es que las rutinas no paralelizadas mantengan la misma performance que en la versión serial.

6. Conclusiones

Se ha implementado en el código de integración explícita de las ecuaciones de movimiento Stampack un esquema de ejecución en paralelo de las tareas. Esta paralelizacion se encuentra basada en el estándar de memoria compartida OpenMP. Esta implementación resulta deseable pues como se muestra en el informe, implica una mínima intervención sobre los archivos fuente del software. Por otra parte esta metodología cuenta con la ventaja de sacar el máximo provecho a los ordenadores actuales del tipo multi-núcleo y multi-tarea (ej.: la nueva familia de procesadores de Intel Core i3, i5, e i7). Estos procesadores se encuentran disponibles actualmente en la mayoría de los ordenadores de escritorio, con lo cual no es necesario contar ordenadores en paralelo.

Actualmente la intervención de los archivo fuentes del código esta circunscrita a las rutinas donde se evalúa el residuo a nivel elemental, y algunas otras áreas asociadas principalmente a la rutina de integración explícita de las ecuaciones de movimiento.

Se han encontrado algunos inconvenientes asociados al manejo de memoria que pue-

de realizar coherentemente cada hilo o thread, y además otro asociados con el atributo SAVE en las variables dentro de las rutinas llamadas desde la zona paralelizada. Estos inconvenientes han sido solucionados, aunque se debería a futuro definir un esquema de división del trabajo hacia cada procesador o hilo del tipo dinámico.

Los resultado muestran que la paralelizacion del código brinda una disminución significativa de los tiempos de cálculos, aun en procesadores multi-núcleo modestos (ej.: Intel Core 2 Duo). Los tiempos de cálculo entre la ejecución serial y la ejecución en paralelo disminuyen como mínimo un 50 %. Esta disminución no es para nada despreciable considerando los costos, numéricos y económicos, que trae aparejado.

A futuro se debe considerar (además del esquema de división de tareas dinámico) la posibilidad de extender la intervención de los archivos fuentes asociados a los esquemas de contacto, y también mejorar la paralelizacion de la rutina de integración explícita. Estos tres puntos: (a) cálculo de residuo a nivel elemental, (b) integración explícita de las ecuaciones de movimiento, y (c) evaluación de las fuerzas de contacto; son los que concentran aproximadamente el 98 % del tiempo total de análisis en una simulacion típica de estampado o conformado de metales.